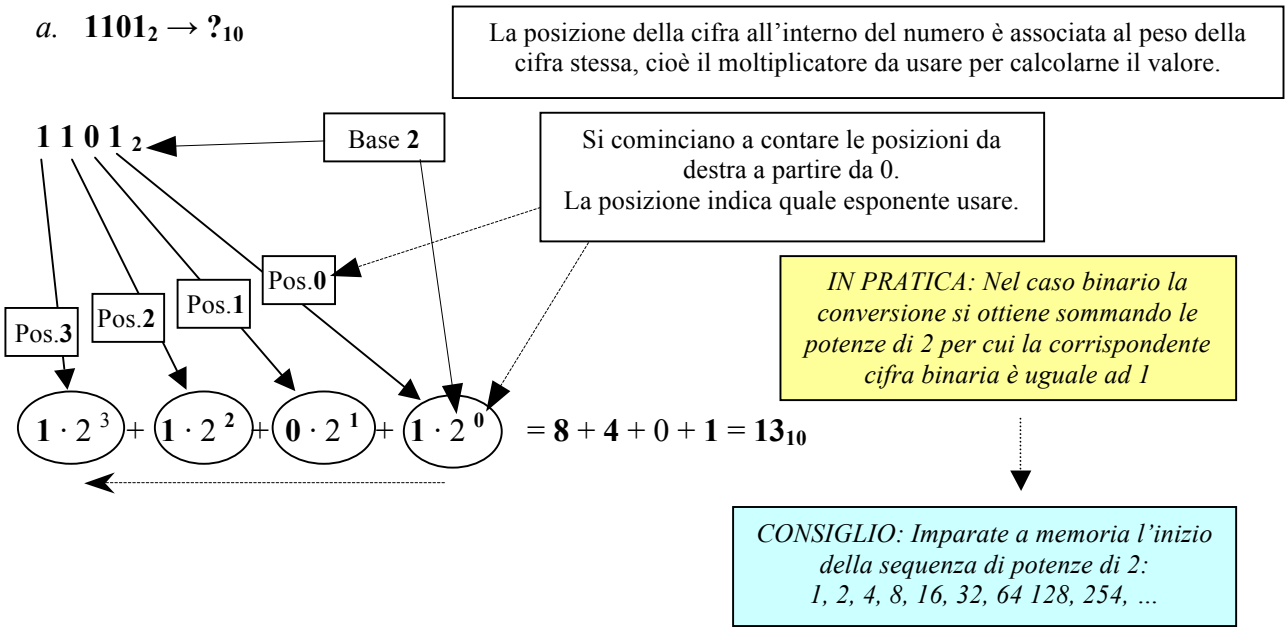
Esercitazione 1 del 8/10/2014

1. Conversione binario → decimale

a. **$1101_2 \rightarrow ?_{10}$**



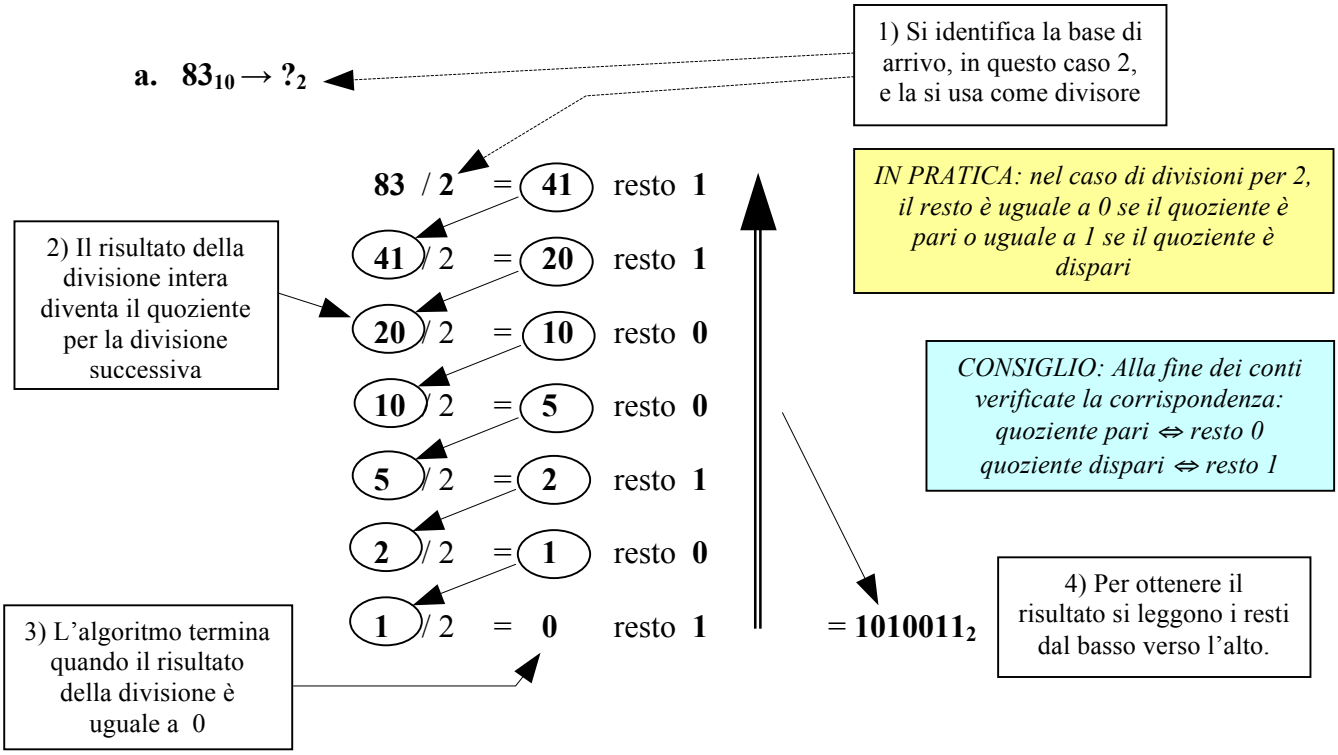
b. **1111_2** $= \mathbf{1} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0$
$= 8 + 4 + 2 + 1 = \mathbf{15_{10}}$

c. **1000011_2** $= \mathbf{1} \cdot 2^6 + \mathbf{0} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0$
$= 64 + 2 + 1 = \mathbf{67_{10}}$

d. **10001001_2** $= \mathbf{1} \cdot 2^7 + \mathbf{0} \cdot 2^6 + \mathbf{0} \cdot 2^5 + \mathbf{1} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0$
$= 128 + 8 + 1 = \mathbf{137_{10}}$

Massimo Marchi – marchi@di.unimi.it

2. Conversione decimale → binario



b. $101_{10} \rightarrow ?_2$

101	/ **2** =	**50**	resto	**1**
50	/ **2** =	**25**	resto	**0**
25	/ **2** =	**12**	resto	**1**
12	/ **2** =	**6**	resto	**0**
6	/ **2** =	**3**	resto	**0**
3	/ **2** =	**1**	resto	**1**
1	/ **2** =	**0**	resto	**1**

$\mathbf{101_{10} = 1100101_2}$

c. $3032_{10} \rightarrow ?_2$

3032	/ **2** =	**1516**	resto	**0**
1516	/ **2** =	**758**	resto	**0**
758	/ **2** =	**379**	resto	**0**
379	/ **2** =	**189**	resto	**1**
189	/ **2** =	**94**	resto	**1**
94	/ **2** =	**47**	resto	**0**
47	/ **2** =	**23**	resto	**1**
23	/ **2** =	**11**	resto	**1**
11	/ **2** =	**5**	resto	**1**
5	/ **2** =	**2**	resto	**1**
2	/ **2** =	**1**	resto	**0**
1	/ **2** =	**0**	resto	**1**

$\mathbf{3032_{10} = 101111011000_2}$

d. $5454_{10} \rightarrow ?_2$

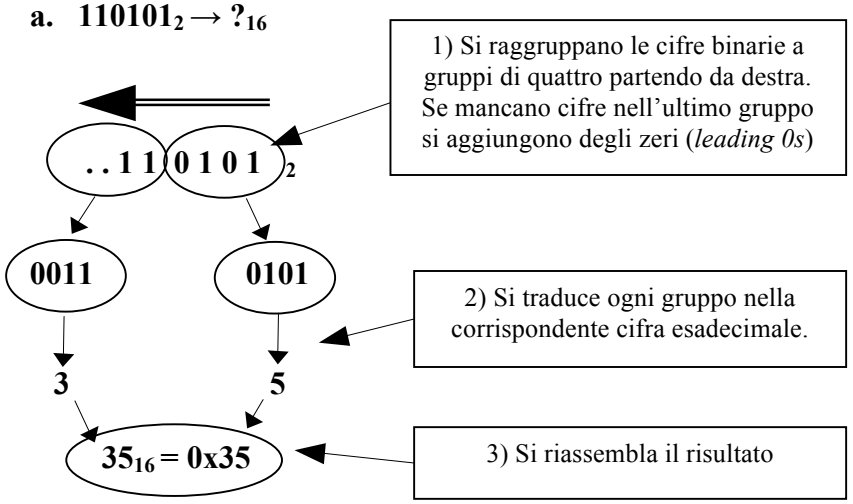
5454	/ **2** =	**2727**	resto	**0**
2727	/ **2** =	**1363**	resto	**1**
1363	/ **2** =	**681**	resto	**1**
681	/ **2** =	**340**	resto	**1**
340	/ **2** =	**170**	resto	**0**
170	/ **2** =	**85**	resto	**0**
85	/ **2** =	**42**	resto	**1**
42	/ **2** =	**21**	resto	**0**
21	/ **2** =	**10**	resto	**1**
10	/ **2** =	**5**	resto	**0**
5	/ **2** =	**2**	resto	**1**
2	/ **2** =	**1**	resto	**0**
1	/ **2** =	**0**	resto	**1**

$\mathbf{5454_{10} = 1010101001110_2}$

Massimo Marchi – marchi@di.unimi.it

3. **Conversione binario → esadecimale**

a. $\mathbf{110101_2 \rightarrow ?_{16}}$



CONSIGLIO: Poiché le possibili combinazioni (16) sono poche, conviene imparare a memoria la tabella lookup di conversione

Tabella lookup

0000 ↔ **0**	1000 ↔ **8**
0001 ↔ **1**	1001 ↔ **9**
0010 ↔ **2**	1010 ↔ **A**
0011 ↔ **3**	1011 ↔ **B**
0100 ↔ **4**	1100 ↔ **C**
0101 ↔ **5**	1101 ↔ **D**
0110 ↔ **6**	1110 ↔ **E**
0111 ↔ **7**	1111 ↔ **F**

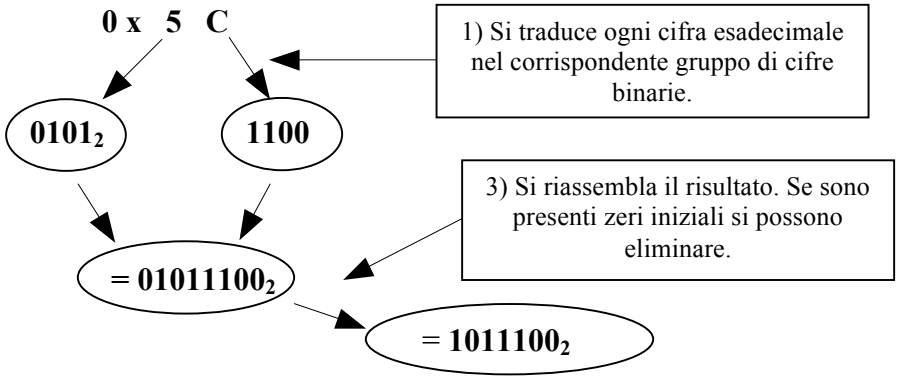
b. $\mathbf{110\ 1101_2} = \mathit{0}110_2 \mid 1101_2 = 0x6 \mid 0xD = \mathbf{0x6D}$

c. $\mathbf{110010011_2} = \mathit{000}1_2 \mid 1001_2 \mid 0011_2 = 0x1 \mid 0x9 \mid 0x3 = \mathbf{0x193}$

d. $\mathbf{111001001010_2} = 1110_2 \mid 0100_2 \mid 1010_2 = 0xE \mid 0x4 \mid 0xA = \mathbf{0xE4A}$

4. **Conversione esadecimale → binario**

a. $\mathbf{0x5C \rightarrow ?_2}$



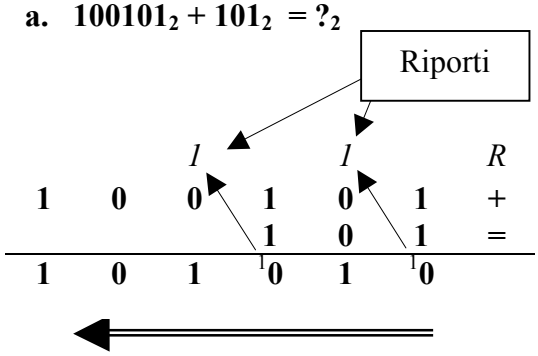
b. $\mathbf{0xA51} = 0xA \mid 0x5 \mid 0x1 = 1010_2 \mid 0101_2 \mid 0001_2 = \mathbf{101001010001_2}$

c. $\mathbf{0x746} = 0x7 \mid 0x4 \mid 0x6 = \cancel{0}111_2 \mid 0100_2 \mid 0110_2 = \mathbf{11101000110_2}$

d. $\mathbf{0xB214} = 0xB \mid 0x2 \mid 0x1 \mid 0x4 = 1011_2 \mid 0010_2 \mid 0001_2 \mid 0100_2 = \mathbf{1011001000010100_2}$

5. Somme binarie

a. $100101_2 + 101_2 = ?_2$



La somma binaria si esegue esattamente come quella decimale classica. Si allineano a destra i numeri da sommare quindi si procede sommando ogni coppia di bit e l'eventuale riporto, da destra verso sinistra. I riporti si sommano sulla coppia di bit successiva a sinistra.

In base 2, occorre ricordarsi che:
1 + **1** = **0** riporto **1**
1 + **1** + **1** = **1** riporto **1**

$100101_2 + 101_2 = 101010_2$

CONSIGLIO: verificate sempre i risultati traducendo i numeri binari in decimale e rifacendo i calcoli in questa base

b. $100100_2 + 1010011_2 = 1110111_2$

								R
		1	0	0	1	0	0	+
	1	0	1	0	0	1	1	=
	1	1	1	0	1	1	1	

c. $10000110_2 + 1001001111_2 = 1011010101_2$

					1	*1*	*1*			*R*
		1	0	0	0	0	1	1	0	+
1	0	0	1	0	0	1	1	1	1	=
1	0	1	1	0	1	0	1	0	1	

d. $111001010_2 + 10101001_2 = 1001110011_2$

1	*1*				*1*					*R*
	1	1	1	0	0	1	0	1	0	+
		1	0	1	0	1	0	0	1	=
1	0	0	1	1	1	0	0	1	1	

Massimo Marchi – marchi@di.unimi.it

6. Sottrazioni binarie (in complemento a due)

a. $1001_2 - 110_2 = ?_2$

COMPLEMENTO A DUE

Completamento

1) Estendo le cifre alla rappresentazione scelta se necessario

La sottrazione in **complemento a due** si esegue sommando al primo termine il complemento a due del secondo termine.

Il **complemento a due** si calcola invertendo le cifre bit a bit e quindi sommando 1.

I calcoli si eseguono sul numero di bit della rappresentazione richiesta, o, se non è data nessuna lunghezza, sul numero di cifre del più grande dei due termini più un bit di segno. Se uno dei due termini risulta più corto allora occorre estendere il segno fino alla lunghezza necessaria.

$1001_2 - 110_2$

$\textcircled{0}1001_2 - \textcircled{0}\textcircled{0}110_2$

Bit di segno

2) Calcolo il complemento a due del secondo termine invertendo i bit e sommando uno

$\textcircled{1}1001_2$

S			*1*		*R*
1	**1**	**0**	**0**	**1**	+
0	**0**	**0**	**0**	**1**	=
1	**1**	**0**	**1**	^{1}**0**	+

$\rightarrow$ **-110_2** *in formato* CA2

Il CA2 di un numero negativo espresso in CA2 è il modulo del numero stesso. Ne segue che dato un numero negativo espresso in CA2 si può tornare alla notazione *Modulo&Segno* ricalcolando il CA2 sul numero negativo stesso in modo da estrarre il suo modulo. Es.:
-6 = 11010 (CA2 su 4bit+segno)
CA2(11010)=00101+1=00110=6

$01001_2 + 11010_2$

3) Sommo il primo termine con il complemento a due del secondo.

1	*1*				*R*
	0	**1**	**0**	**0**	**1**	+
	1	**1**	**0**	**1**	**0**	=
~~^{1}~~	^{1}**0**	^{1}**0**	**0**	**1**	**1**	+

$\rightarrow$ **$+11_2$**

Il riporto oltre il bit di segno viene scartato

Può capitare che il risultato sia troppo grande, in modulo, per essere rappresentato dal numero di bit disponibili. In questo caso si dice che si è verificato un errore di *OVERFLOW*.

Ex: Calcolare 3+3 usando 2 bit + segno
3 + 3 = **0**11_2 + **0**11_2 = **1**10_2 = **-2 in CA2 !!**

Ex: Calcolare -2-3 usando 2 bit + segno
-2 - 3 = **1**10_2 + **1**01_2 = **0**01_2 = +1 !!

La condizione di overflow si verifica controllando la coerenza tra segno dei termini sommati ed il risultato.

"+" + "+" = "-" incoerente!
"-" + "-" = "+" incoerente!
"-" + "+" non si verifica mai overflow
"+" + "-" non si verifica mai overflow

Risultato: $1001_2 - 110_2 = +11_2$

*APPROFONDIMENTO: per sottrarre 1 in binario potete cercare il primo **1** a destra, porlo a **0** e quindi porre a **1** tutti gli **0** a destra dell'**1** trovato.*
Ex: $11000_2 - 1_2 = 10111_2$

Massimo Marchi – marchi@di.unimi.it

b. $1001_2 - 10010_2 = ?_2$

Uso cinque cifre (il secondo termine è il più lungo) più il bit di segno:

$1001_2 - 10010_2 = 0\ 01001_2 - 0\ 10010_2 = 001001_2 + (\ 101101_2 + 1\ )$

Calcolo il CA2 di **-1010_2**:

S				*I*		*R*	
1	**0**	**1**	**1**	**0**	**1**	+	
0	**0**	**0**	**0**	**0**	**1**	=	
1	**0**	**1**	**1**	**1**	^{1}**0**		→ **-10010_2** in CA2

Eseguo la somma tra il primo termine e il CA2 del secondo:

S	*I*					*R*	
0	**0**	**1**	**0**	**0**	**1**	+	
1	**0**	**1**	**1**	**1**	**0**	=	
1	**1**	^{1}**0**	**1**	**1**	**1**		→ **-1001_2** in CA2
							(110111→001000+1=1001)

Risultato: **$1001_2 - 10010_2 = -1001_2$**

c. $11111_2 - 1101_2 = ?_2$

Uso cinque cifre (il primo è il più lungo) più il bit di segno:

$11111_2 - 1101_2 = 0\ 11111_2 - 0\ 01101_2 = 011111_2 + (\ 110010_2 + 1\ )$

Calcolo il CA2 di **-1001_2** :

S						*R*	
1	**1**	**0**	**0**	**1**	**0**	+	
0	**0**	**0**	**0**	**0**	**1**	=	
1	**1**	**0**	**0**	**1**	**1**		→ **-1101_2** in CA2

Eseguo la somma tra il primo termine e il CA2 del secondo:

1	*1*	*1*	*1*	*1*	*1*		*R*	
	0	**1**	**1**	**1**	**1**	**1**	+	
	1	**1**	**0**	**0**	**1**	**1**	=	
~~1~~	^{1}**0**	^{1}**1**	^{1}**0**	^{1}**0**	^{1}**1**	^{1}**0**		→ **$+10010_2$**

Risultato: **$11111_2 - 1101_2 = +10010_2$**

Massimo Marchi – marchi@di.unimi.it

d. $\mathbf{10011_2 - 101101_2 = ?_2}$ (*Eseguire i calcoli a 8 bit)*

Uso sette cifre più il bit di segno:

$\mathbf{10011_2 - 101101_2 = 0\ 0010011_2 - 0\ 0101101_2 = 00001110_2 + (\ 11010010_2 + 1_2\ )}$

S							*R*	
1	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**+**	
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**=**	
1	**1**	**0**	**1**	**0**	**0**	**1**	**1**		→ $\mathbf{-101101_2}$ in CA2

Eseguo la somma tra il primo termine e il CA2 del secondo:

S		*1*			*1*	*1*		*R*	
0	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**+**	
1	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**=**	
1	**1**	**1**	^{1}**0**	**0**	**1**	^{1}**1**	^{1}**0**		→ $\mathbf{-11010_2}$ in CA2
						(11100110→00011001+1=11010)			

Risultato: $\mathbf{10011_2 - 101101_2 = -11010_2}$

7. Conversione in floating point secondo lo standard IEEE 754

a. $\mathbf{-20{,}75_{10}} = \langle s,e,m \rangle$?

> Per convertire in floating point occorre:
> - calcolare il segno
> - convertire la parte intera in binario (es: con il metodo delle divisioni per 2 successive)
> - convertire la parte frazionaria in binario (es: con il metodo delle moltiplicazioni per 2 successive)
> - unire i due risultati e normalizzare.
> - calcolare la mantissa secondo la precisione voluta (occorre scartare il primo 1 della normalizzazione)
> - calcolare l'esponente della normalizzazione polarizzato e convertirlo in binario secondo la precisione voluta

Numero negativo: $\boldsymbol{s = 1}$

Converto 20_{10} in base 2: $\mathbf{20_{10} = 10100_2}$

20	**/ 2**	=	**10**	resto **0**
10	**/ 2**	=	**5**	resto **0**
5	**/ 2**	=	**2**	resto **1**
2	**/ 2**	=	**1**	resto **0**
1	**/ 2**	=	**0**	resto **1**

Converto $0{,}5_{10}$ in base 2: $\mathbf{0{,}75_{10} = 0{,}11_2}$

> La conversione della parte frazionaria **per moltiplicazioni per 2 successive** si esegue nel seguente modo:
> - si moltiplica per 2 la parte frazionaria. La parte intera del risultato costituisce il primo bit della parte frazionaria espressa in binario.
> - Si ripete il passo precedente sulla parte frazionaria del risultato. La parte intera del risultato costituirà adesso il secondo bit della parte frazionaria espressa in binario.
> - Si ripete il procedimento ricavando i successivi bit fino a che la parte frazionaria risulta uguale a zero (tutti i bit successivi saranno a zero) oppure si è raggiunta la precisione voluta (es. si sono ricavati già i 23 bit necessari per una mantissa in precisione singola).

NOTA PER IL CORSO: l'IEEE754 prevede anche un algoritmo per calcolare l'ultimo bit della mantissa per arrotondamento dei successivi in modo da migliorare la rappresentazione ottenuta. Per i fini del corso sarà sufficiente troncare la mantissa al 23-esimo bit ignorando l'arrotondamento eventuale.

0,75	*** 2**	=	**1,5**	parte intera **1**
0,5	*** 2**	=	**1,0**	parte intera **1**
#######				

Leggo i bit dall'alto verso il basso. **Il bit più vicino alla virgola è il primo bit trovato.**

La parte frazionaria è uguale a zero. Il metodo termina.

Unisco i risultati: $\mathbf{20{,}75_{10} = 10100{,}11_2}$

Normalizzo: $\mathbf{10100{,}11_2 = 1{,}010011_2 \cdot (10_2)^4}$

Mantissa a 23 bits: $\mathbf{1{,}010011_2 \rightarrow m = 0100110000\ 0000000000\ 000}$

Polarizzo l'esponente: $\mathbf{4_{10} + 127_{10} = 131_{10} = 128_{10} + 2_{10} + 1_{10} = 10000011_2}$

Esponente a 8 bits: $\mathbf{10000011_2 \rightarrow e = 10000011}$

$\mathbf{-20{,}75_{10} = \langle 1,\ 10000011,\ 01001100000000000000000 \rangle}$

Massimo Marchi – marchi@di.unimi.it

b. $17{,}375_{10}$ = <s,m,e>?

Numero positivo: $s = 0$
Converto 17_{10} in base 2: $17_{10} = 10001_2$ $(17=16+1=10000_2+1_2=10001_2)$
Converto $,375_{10}$ in base 2: $0{,}375_{10} = 0{,}011_2$

0,375	*** 2**	=	**0,75**	parte intera **0**
0,75	*** 2**	=	**1,5**	parte intera **1**
0,5	*** 2**	=	**1,0**	parte intera **1**

#########

Unisco i risultati: $17{,}375_{10} = 10001{,}011_2$
Normalizzo: $10001{,}011_2 = 1{,}0001011_2 \cdot (10_2)^4$ *(shift a sinistra di 4 posizioni: exp= 4)*
Mantissa a 23 bit: ~~1~~$,0001011_2 \rightarrow m = 00010\ 11000\ 00000\ 00000\ 000$
Polarizzo l'esponente: $4_{10} + 127_{10} = 131_{10} = 10000011_2$

131	**/ 2**	=	**65**	resto **1**
65	**/ 2**	=	**32**	resto **1**
32	**/ 2**	=	**16**	resto **0**
16	**/ 2**	=	**8**	resto **0**
8	**/ 2**	=	**4**	resto **0**
4	**/ 2**	=	**2**	resto **0**
2	**/ 2**	=	**1**	resto **0**
1	**/ 2**	=	**0**	resto **1**

Esponente a 8 bits: $10000011_2 \rightarrow e = 10000011$

$17{,}375_{10}$ = <0, 10000011, 00010110000000000000000>

c. $-0{,}78125_{10}$ = <s,m,e>?

Numero negativo: ***s* = 1**
Converto 0_{10} in base 2: $\mathbf{0_{10} = 0_2}$

Converto $\cdot{,}4375_{10}$ in base 2: $\mathbf{0{,}78125_{10} = 0{,}11001_2}$

0,78125 * 2 = 1,5625 parte intera **1**
0,5625 * 2 = 1,125 parte intera **1**
0,125 * 2 = 0,25 parte intera **0**
0,25 * 2 = 0,5 parte intera **0**
0,5 * 2 = 1,0 parte intera **1**
#########

Unisco i risultati: $\mathbf{0{,}78125_{10} = 0{,}11001_2}$
Normalizzo: $\mathbf{0{,}11001_2 = 1{,}1001_2 \cdot (10_2)^{-1}}$ *(shift a destra di 1 posizioni: exp=-1)*
Mantissa a 23 bit: $\mathbf{\cancel{1},1001_2 \rightarrow m = 1001000000\ 0000000000\ 000}$
Polarizzo l'esponente: $\mathbf{-1_{10} + 127_{10} = 126_{10} = 1111110_2}$

126 / 2 = 63 resto **0**
63 / 2 = 31 resto **1**
31 / 2 = 15 resto **1**
15 / 2 = 7 resto **1**
7 / 2 = 3 resto **1**
3 / 2 = 1 resto **1**
1 / 2 = 0 resto **1**

Esponente a 8 bits: $\mathbf{1111110_2 \rightarrow e = 01111110}$

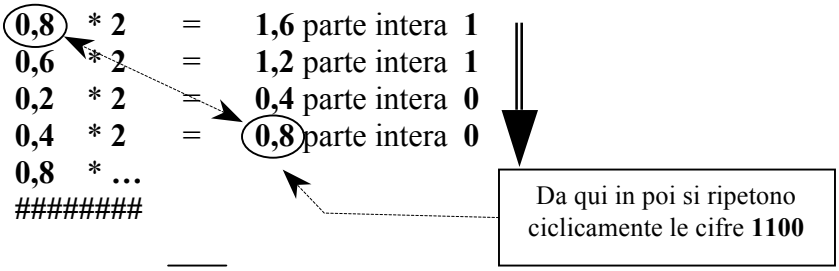
$0{,}78125_{10}$ <0, 01111110, 10010000000000000000>

d. $-0{,}8_{10}$ = <s,m,e>?

Numero negativo: ***s* = 1**

Converto 0_{10} in base 2: $\mathbf{0_{10} = 0_2}$

Converto $\cdot{,}3_{10}$ in base 2: $\mathbf{0{,}8_{10} = 0, \overline{1100}_2}$



Unisco i risultati: $\mathbf{0{,}8_{10} = 0, \overline{1100}_2}$

Normalizzo: $\mathbf{0, \overline{1100}_2 = 0, 1100\ \overline{1100}_2 = 1{,}100\ \overline{1100}_2 \cdot (10_2)^{-1}}$

Mantissa a 23 bit: $\mathbf{\cancel{1}, 100\ 1100\ 1100\ 1100\ 1100\ 1100\ \cancel{1100}}$

$\mathbf{\rightarrow m = 10011001100110011001100}$

Tronco la rappresentazione periodica della mantissa alla lunghezza fissa di 23 bit

Polarizzo l'esponente: $\mathbf{-1_{10} + 127_{10} = 126_{10} = 1111110_2}$

126	**/ 2 =**	**63**	resto	**0**
63	**/ 2 =**	**31**	resto	**1**
31	**/ 2 =**	**15**	resto	**1**
15	**/ 2 =**	**7**	resto	**1**
7	**/ 2 =**	**3**	resto	**1**
3	**/ 2 =**	**1**	resto	**1**
1	**/ 2 =**	**0**	resto	**1**

NOTA PER IL CORSO: l'IEEE754 calcola l'ultimo bit della mantissa per arrotondamento dei successivi in modo da migliorare la rappresentazione ottenuta. In questo esempio considerando anche l'arrotondamento, l'ultimo bit risulterebbe uguale a 1. Per i fini del corso è sufficiente troncare la mantissa ottenuta al 23-esimo bit ignorando l'arrotondamento.

Esponente a 8 bits: $\mathbf{1111110_2 \rightarrow e = 01111110}$

$-0{,}8_{10}$ = <1, 01111110, 10011001100110011001100 >

Alcune configurazioni con significato speciale (singola precisione):

0 <0,00000000,00000000000000000000000>

+/-∞ <[*segno*],11111111,00000000000000000000000>

NaN <[*segno*],11111111, [*mantissa* ≠ *0*]>

N.ro denormalizzato <[*segno*],00000000, [*mantissa denormalizzata≠0*]> *(vedi di seguito)*

Alcuni casi notevoli (singola precisione):

1 = $1{,}0 \cdot (10_2)^0$ <0,01111111,00000000000000000000000> *(E=0+127)*

MaxFloat = $1{,}11..1 \times 2^{+127}$ <0,11111110,11111111111111111111111> *(E=+127+127)*

MinFloat = $1{,}0 \times 2^{-126}$ <0,00000001,00000000000000000000000> *(E=-126+127)*

MinFloatDeN = $0{,}0..01 \times 2^{-126}$ <0,00000000,00000000000000000000001>

saperne di più: ***http://stevehollasch.com/cgindex/coding/ieeefloat.html***

Massimo Marchi – marchi@di.unimi.it